I) Définitions

Class: Un composant abstrait contenant des attributs et des méthodes et permettant de créer des objets.

Objet: Instance d'une classe qui affecte des valeurs aux attributs de la classe.

Exemple:

Personne
+ age + nom
marcher ( ) manger ( )

} classe
} attributs
} méthodes

instanciation → age = "50", nom = "Taoussi" — objet 1 de la class personne

instanciation → age = "40", nom = "Zaki" — objet 2 de la class personne

La création des objets à partir d'une classe s'appelle l'instanciation et se fait via des méthodes spéciale s'appelle les constructeurs.

Instanciation: c'est l'operation de création d'un objet à partir d'une classe en utilisant le mot réservé new.

Constructeurs: Sont des méthodes spéciales qui font partie de la classe et qui servant à créer des objets.

les constructeurs doivent s'aligner avec deux règles:

1/ Ils portent le même nom de la classe.

2/ Ne retourne rien et ne contient pas void.

Exemple:

```
Class personne {
String nom;
 Personne (string nom) {
  this.nom = nom;
 }
}
```

} le constructeur:
* même nom que la classe
* ne retourne rien
* ne contient pas void

this: permet de faire la différence entre l'attribut de la classe et le paramètre du constructeur car ils prenent le même nom.

this() : permet de faire des appels entre les constructeurs de la même classe.

this(100) : appelle le constructeur qui prend int

this("A") : appelle le constructeur qui prend string.

Exemple :

```
class Ecole {
  Directeur directeur;
  string nom;
  Ecole(Directeur directeur, string nom){
    this("nom");
    this.directeur = directeur;
    Sop(this.nom);
  }
  Ecole(Directeur directeur){
    this.nom = directeur.nom;
  }
  Ecole(string nom) {
    this(new directeur("Ahmed"));
    Sop(nom);
  }
}
```

```
Class Directeur {
  int age;
  string nom;
  Directeur(string nom){
    this(100);
    Sop(nom);
  }
  Directeur(int age){
    this.age = age++ + ++age;
    Sop(this.age);
  }
}
```

```
Directeur d = new Directeur("A");   → 202
                                      A
Ecole e = new Ecole(d, "B");        → 202
                                      Ahmed
                                      Nom
                                      NULL
```

II - Encapsulation :

Encapsulation : C'est un principe orienté objet qui consiste à verouiller les attributs d'une classe et créer deux méthodes :

a) Une méthode pour la consultatation (Getter) :

Cette méthode permet de consulter la valeur stocker dans un attribut.

Exemple :

```
class Etudiant {
  private string nom;
  public string getNom() {
    return this.nom;
  }
}
```

règles :

1/ le Getter d'un attribut est public.

2/ le Getter d'un attribut retourne le même type que l'attribut.

3/ le Getter d'un attribut Commence par "get"

b) Une méthode pour la modification (Setter):

Cette méthode permet de modifier la valeur d'un attribut par une nouvelle valeur.

Exemple:
```
class Etudiant {
    private string nom;
    Public Void SetNom (string nom){
    this.nom = nom;
    }
}
```

Règles:

1/ le Setter d'un attribut est public.

2/ le setter d'un attribut modifie sans aucun retourn

3/ le setter d'un attribut commence par "set".

Exercice:

Créer une classe Matrice définie par id (string), libelle (string), le nombre d'heure (int) et validation (boolean).

1) Créer les attributs de cette classe.

2) Créer deux constructeurs:

- le premier prend l'id; le libelle, le nombre d'heure et validation.
- le deuxième prend l'id et validation.

3) Créer les Getters et setters des attributs.

Solutions:
```
class Matière {
  Private string id;
  private string libelle;
  private int nbrheure;
  private boolean valid;

public Matière (string id, string libelle, int nbrheure, boolean valid){
    this.id = id;
```

```
    this.libelle = libelle;
    this.nbrheure = nbrheure;
    this.valid = valid;
  }

public Matiere(string id, boolean valid){
    this.id = id;
    this.valid = valid;
}

public string getId() { return this.id; }
public string getLibelle() { return this.libelle; }
public int getNbrheure() { return this.nbrheure; }
public boolean getValid() { return this.valid; }
public void setId(string id) { this.id = id; }
public void setLibelle(string libelle) { this.libelle = libelle; }
public void setNbrheure(int nbrheure) { this.nbrheure = nbrheure; }
public void setValid(boolean valid) { this.valid = valid; }
}
```

III - Variables de classe et variables d'objets :

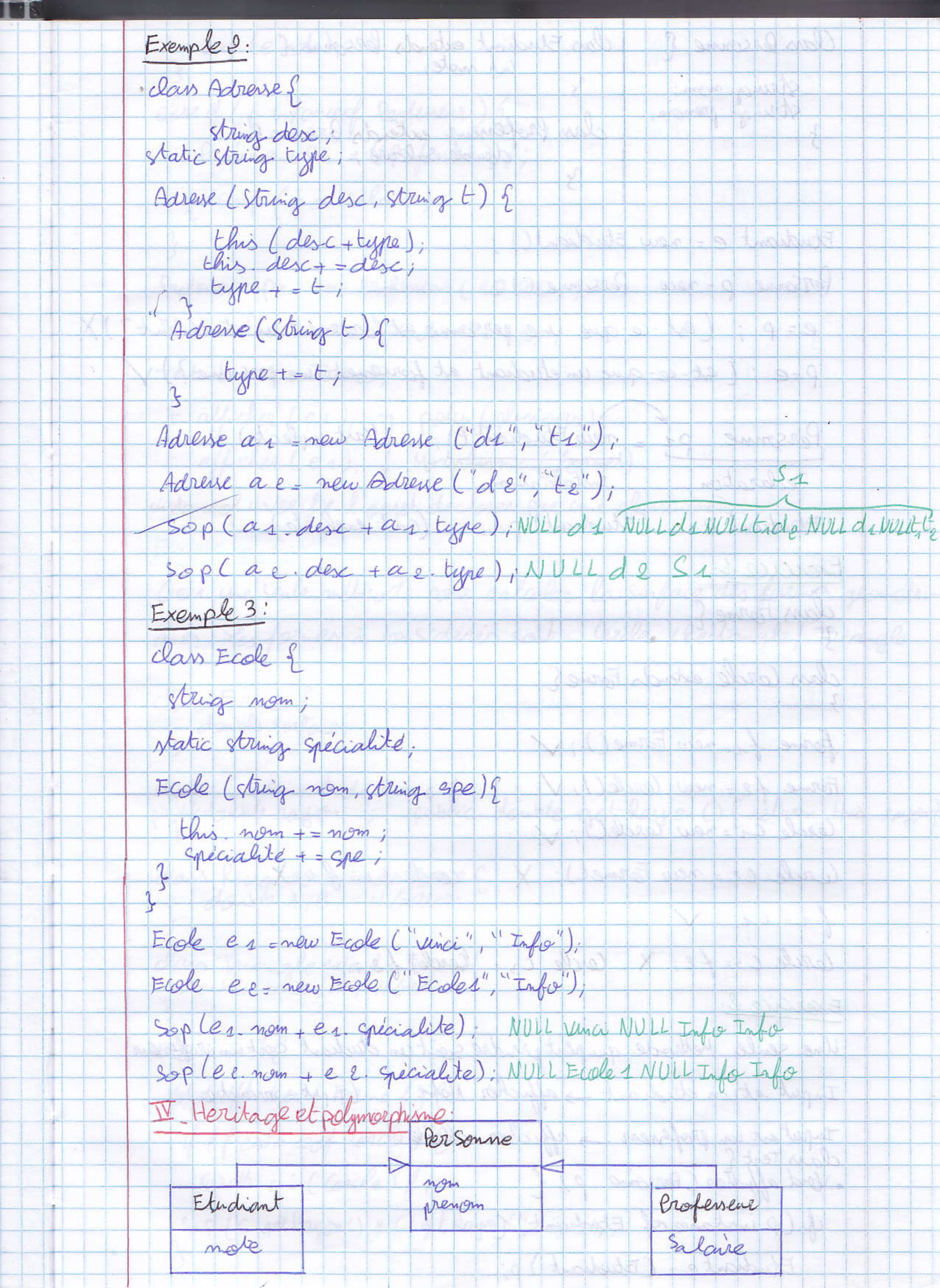
Exemple 1 :

```
Class Etudiant {
    public String nom;
    private int age;
    static int nbEtudiants;

  public Etudiant(String nom, int age, int nb) {
      this.nom = nom;
      this.age = age;
      nbEtudiants = nb;
  }

Etudiant e1 = new Etudiant("Ahmed", 20, 20);
 Sop(e1.nom + e1.age + e1.nbEtudiants);        Ahmed 20 20
Etudiant e2 = new Etudiant("Omar", 30, 25);
 Sop(e1.nom + e1.age + e1.nbEtudiants);        Ahmed 20 25
 Sop(e2.nom + e2.age + e2.nbEtudiants);        Omar 30 25

                              Etudiant.nbEtudiant
```



Exemple 2 :

```
class Adresse {
      string desc ;
static string type ;
 Adresse (String desc, string t) {
      this (desc + type) ;
      this . desc + = desc ;
      type + = t ;
   }
 Adresse (String t) {
      type + = t ;
 }
```

```
Adresse a1 = new Adresse ("d1", "t1") ;
Adresse a2 = new Adresse ("d2", "t2") ;
Sop ( a1 . desc + a1 . type ) ; NULL d1 NULL d1 NULL t1 d2 NULL d1 NULL t1 t2
sop ( a2 . desc + a2 . type ) ; NULL d2 S1
```

S1 = NULL d1 NULL t1 d2 NULL d1 NULL t1 t2

Exemple 3 :

```
class Ecole {
 string nom ;
 static string spécialite ;
 Ecole (string nom, string spe) {
   this . nom + = nom ;
   spécialite + = spe ;
 }
}
```

```
Ecole e1 = new Ecole ("vinci", "Info") ;
Ecole e2 = new Ecole ("Ecole1", "Info") ;
Sop (e1 . nom + e1 . spécialite) ;   NULL vinci NULL Info Info
sop (e2 . nom + e2 . spécialite) ;   NULL Ecole 1 NULL Info Info
```

IV _ Heritage et polymorphisme :

```
Class Personne {
    string nom;
    string prenom;
}

class Etudiant extends Personne {
    int note;
}

class Professeur extends Personne {
    double salaire;
}
```

```
Etudiant e = new Etudiant() ;
Personne p = new Personne () ;
e = p ; (est-ce-que une personne est forcement un etudiant ?) X
p = e ; (est-ce-que un etudiant est forcement une personne ?) ✓

Personne p1 = new Etudiant () ;      (cast implecite)
Declaration   Instanciation

Etudiant e1 = (Etudiant) p1 ;     (Cast explicite)
```

Exercice 1:

```
class Forme {
}
class Cercle extends Forme {
}

Forme f1 = new Forme () ; ✓
Forme f2 = new Cercle(), ✓
Cercle c1 = new Cercle(); ✓
Cercle c2 = new Forme(); X      cercle c2 = f1; X
f1 = c1;   ✓
Cercle c3 = f2; X   Cercle c3 = (Cercle) f2;
```

Exercice 2:

Une seule Methode qui peut prendre soit un etudiant soit un professeur.

Input est un etudiant → afficher Note.   (Input = paramètre)

Input est un professeur → afficher salaire.

```
class Test {
• void afficher (Personne p) {
   if (p instance of Etudiant) {
      Etudiant e = (Etudiant) p;
```

```
    Sop (e.note);
  }
else if (p instanceof Professeur) {
    Professeur p1 = (Professeur) p;
      Sop (p1. salaire);
   }
 }
```

```
Professeur p1 = new Professeur (10 000);
Etudiant e1 = new Etudiant (19);
Test t = new Test ();
 t. affichier (p1);
 t. affichier (e1);
```

poly (plusieurs) Morphisme (formes)

instanceof : verifie l'objet déclarer par une classe mère.

Exercice 3 :

Créer une seule méthode pour calculer la surface des formes géométriques. Les formes à considérés sont : Cercle / Rectangle / Triangle.

```
class Forme {
    double surface;
 }

class Cercle extends Forme {
private double rayon    ;    public double getRayon () { return this. rayon;}
 }

class Rectangle extends Forme {
    double A, B;   // Getters
 }

class Triangle extends Forme {
  double base, h; // Getters
 }

class Test {
 Void affichier ( Forme f) {
  if (f instanceof Cercle) {
    Cercle c = (Cercle) f;
  Sop (c. getRayon() * c. getRayon () * 3.14);
 }
```

```
else if (f instanceof Rectangle) {
  Rectangle r = (rectangle) f ;
  Sop ( r.getA() * r.getB());
}
else if (f instanceof Triangle) {
Triangle t = (Triangle) f ;
 Sop( getBase() * getH() / 2 ; }
}
 }
class A {   Test t = new Test() ;
  Forme f = new Forme ();
  Triangle t1 = new Triangle ();
  ~~Test~~ t . Calculer (f) => aucun affichage
     t. Calculer (t1) => affiche la formule du Triangle.
```

Exercice 4:

Créer une méthode qui affiche la note des examens.

Si l'examen est pratique la note est sur 100.

Si l'examen est théorique la note est sur 20.

```
Class Examen { }
Class Pratique extends Examen { }
Class Théorique extends Examen { }
class Test {
  Void afficher (Examen e) {
           Examen  Pratique  Théorique
  if ( e instanceof Pratique )
      Sop ("La note est sur 100");
  if ( e instanceof Théorique )
      Sop ("La note est sur 20");
 }
class A {
 Test t = new Test(); ~~B Examen~~
 Examen e = new Examen();
```

```
Pratique p = new Pratique();
Théorie t1 = new Théorique();
t.afficher(e); => Aucun
t.afficher(p); => la note sur 100
t.afficher(t1); => la note sur 20
```

Exercice 5:

```
class Personne {                          //Objet : class mère par défault
  3 -> Personne(){
    4 -> Sop("CONST PERSONNE");
       }
     Personne(int age){ sop("AGE"); }
  }
class Etudiant extends Personne {
        Etudiant(){
      1 -> this(20);
      6 -> Sop("CONST ETD");
        }
2 -> Etudiant(int age){ super(); //constructeur par défault
 5 -> sop("Etudiant AGE");
   }
 }
class Test {
   Etudiant e = new Etudiant();
 }
```

CONST PERSONNE
Etudiant AGE
CONST ETD

Exercice 6:

```
class Forme{
 Forme() {
 3 -> this(20);
    Sop("CONST1");
  }
 Forme(int surface) {
4 -> Sop("CONST2");
 }
```

```
}
class Cercle extends Forme {
   Cercle () {
1 → this("C1");
      Sop ("CONST3");
   }
2 → Cercle (String C){
                           → super()
      Sop(C);
   }
class Test { public static void main (String [] args){
Cercle C1 = new Cercle();
}
```

CONST2

CONST1

C1

CONST3

Super() ⇒ Appel constructeur de la classe mère

Exercice 7:

```
Class Personne {
  Void marcher () {      // Signature
      Sop ("Je suis Personne et je marche normalement"); // CORPS
    }
 }
class Etudiant extends Personne {
}
class Handicape extends Personne {
    Void marcher() {          // Redefinition du methode marcher
        Sop("Je suis handicape");
     }
  }
  Personne p = new Personne();
        p. marcher();   Je suis Personne
  Etudiant e = new Etudiant();
        e. marcher();   Je suis Personne
  Handicape h = new Handicape
```

h. marcher(); Je suis handicape

V_ final

final → Classe (ne peut pas avoir des classe fille)
final → Methode (ne peut pas avoir une Redefinition)
final → Attribut (constante) devrait être initialisée

Exercice 8:

```
class Ecole {
    void former() {
        sop("Ecole");
    }
}
class Commerce extends Ecole {
    void former() {
        sop("Commerce");
    }
    void publier() {
        sop("Publier");
    }
}
Ecole e1 = new Ecole();
Ecole e2 = new Commerce();
Commerce e3 = new Commerce();
e1.former(); ecole
e2.former(); Commerce
e3.former(); Commerce
e1.publier(); Erreur
e2.publier(); Erreur
e3.publier(); publier
```

Exercice 9:

```
class Livre {
    void afficher() {
        sop("A");
    }

    void livre() {
```

```
        sop("B");
    }
}

class Roman extends Livre {
    void afficher(){
        sop("C");
    }
}

Livre l1 = new Livre();
Roman l2 = new Roman();
Livre l3 = new Roman();
Roman l4 = new Livre(); //Erreur : un livre n'est pas forcement un Roman
l1.afficher();   A
l2.afficher();   C
l3.afficher();   C
l4.afficher();   erreur
l1.livre();      B
l2.livre();      B
l3.livre();      B
l4.livre();      erreur
```

1) final (class):

```
final class Ecole {
}
class EcoleFille extends Ecole {
      ↳ erreur de compilation
}
```

2) final (Methode):

```
class Ecole {
  final int add (int a, int b) {return a+b;}
}
class EcoleFille extends Ecole {
  int add (int a, int b) {return 10*(a+b);}
}       ↳ erreur de compilation.
```

3) final(Attribut):

```
class Ecole { final int a1 = 10 ;
                a1 = 20 ; erreur de compilation
}
```

VI _ Abstract:

Exemple:

```
abstract class Animal {                    // ∃ méthode abstraite
                                           => la classe est abstraite
  abstract void marcher();                 <= ??
           // méthode n'a pas de corp
           void manger(){};
}
class Chat extends Animal {
      void marcher(){
         sop("Je suis un chat....");
  }
}
Abstract class AnimalA extends Animal {
}    // la classe est abstraite ⇏ ∃ méthode abstrate
abstract class Chat extends AnimalA {
}
class ChatFille extends chat {
  void marcher() { sop("description");
  }
}
```

Exercice 1:

1) Créer une Classe Forme Géométrique Contenant la méthode abstraite double calculer Surface();

2) Créer deux classes filles de FormeGeometrique : classe Forme-Catégorie A et Forme Catégorie B.

3) Créer Cercle fille de Forme Categorie A défini par un rayon comme attribut => Donner le corps à la méthode Calculer Surface().

4) Créer Triangle fille de Forme Catégorie B définie par la base et la hauteur comme attributs ⇒ Donner le corps à la méthode calculer Surface().

Exercice 2 :

Trouver les erreurs de compilation :

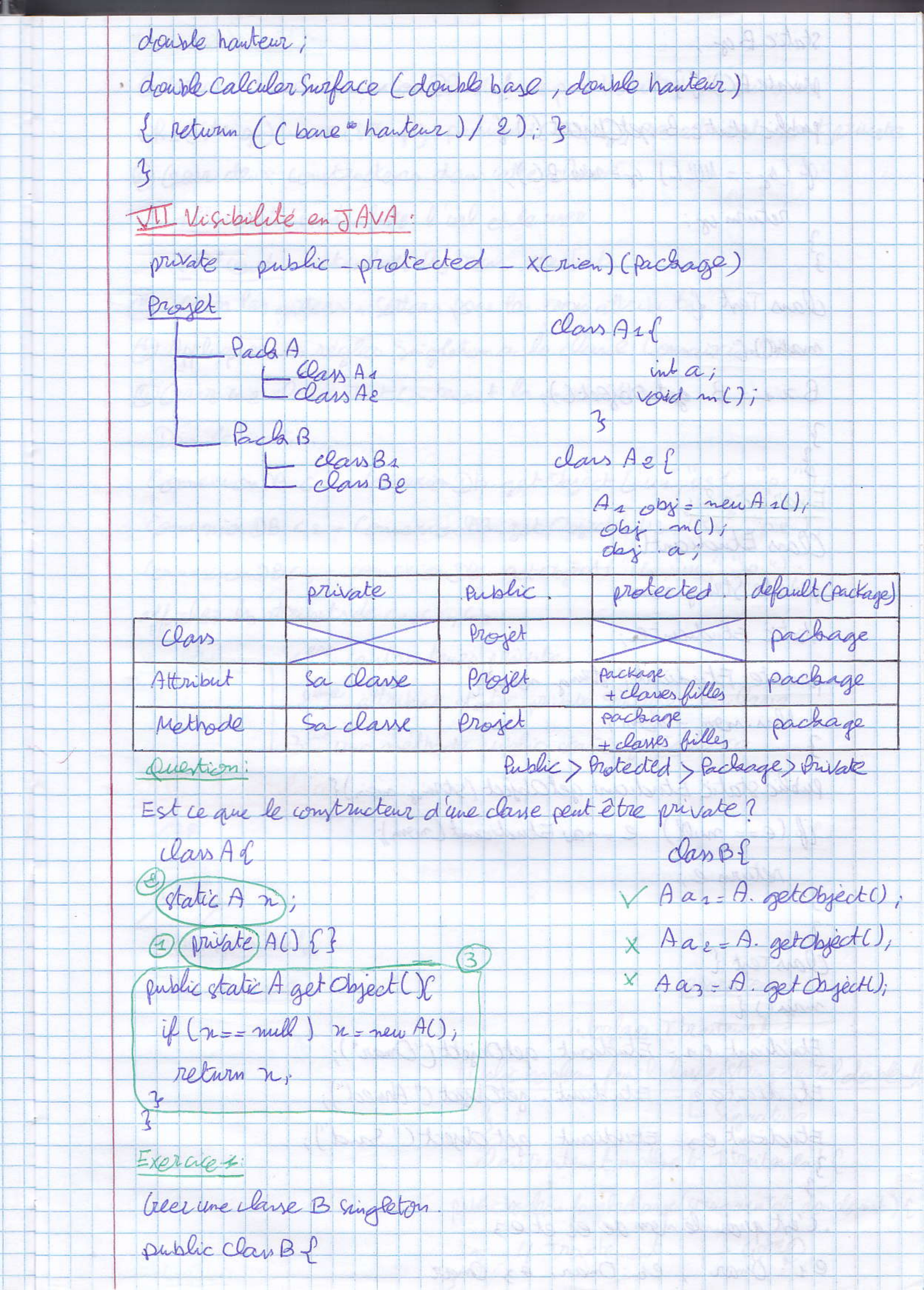
```
abstract class Personne {
abstract void afficher();
  void marcher() {
     Sop("Je suis Personne");
  }
}

Abstract class PersonneA extends Personne {
   void marcher() {
       Sop("Je suis Personne A");
   }
}
class Etudiant extends PersonneA {
        void afficher() {
            Sop("afficher"); }
}
```

solutions :

Exercice 1 :

```
abstract class FormeGeometrique {
  abstract double calculerSurface();
}
abstract class FormeCategorieA extends FormeGeometrique {}
abstract class FormeCategorieB extends FormeGeometrique {}
class Cercle extends Forme Categorie A {
 double rayon ;
 double Calculer Surface (double rayon)
 {return (rayon*rayon*3,14) ;} // sop(rayon*rayon*3,14);
}
class Triangle extends FormeCategorieB {
double base;
```



```
double hauteur ;
double calculerSurface (double base , double hauteur)
{ return ((base * hauteur ) / 2); }
}
```

VII Visibilité en JAVA :

private - public - protected - X(rien) (package)

Projet
- Pack A
  - class A1
  - class A2
- Pack B
  - class B1
  - class B2

```
class A1 {
    int a ;
    void m();
}

class A2 {
    A1 obj = new A1();
    obj.m();
    obj.a ;
```

	private	Public	protected	default (package)
Class	X	Projet	X	package
Attribut	Sa classe	Projet	package + classes filles	package
Methode	Sa classe	Projet	package + classes filles	package

Public > Protected > Package > Private

Question :

Est ce que le constructeur d'une classe peut être private ?

```
class A {
② static A x ;
① private A() {}
③ public static A getObject() {
     if (x == null) x = new A();
     return x;
  }
}
```

```
class B {
✓ A a1 = A.getObject();
x A a2 = A.getObject();
x A a3 = A.getObject();
```

Exercice 1 :

Creer une classe B singleton.

```
public class B {
```

```
static B o ;
private B() { }
public static B getObjekt () {
if (o == NULL) o = new B();
  return o;
}
}
class Test {
main() {
B x1 = B. getObject();
}
}
```

Exercice 2 :

```
Class Etudiant {
private String nom ;
static Etudiant e ;
private Etudiant (String nom) {
  this.nom = nom ;
}
public static Etudiant getObject (String nom) {
if (e == null)  e = new Etudiant (nom);
  return e ;
}
}
Class Test {
main() {
Etudiant e1 = Etudiant. getObjekt ("Omar");
Etudiant e2 = Etudiant. getObject (" Ahmed");
Etudiant e3 = Etudiant. get Object (" Said");
}
}
```

. C'est quoi le nom de e2 et e3

e_1 : Omar , e_2 : Omar, e_3 : Omar

Exercice Singleton:

- Créer une classe Connexion DB définie par les attributs suivants:

URL (String), user (String), password (String). les attributs sont private

(1) Créer deux constructeurs dans cette classe.

1ère const: initialiser l'url et le user

2ème const: initialiser les trois attributs

(2) Créer les getters + setters pour les trois attributs

(3) Appliquer les règles Singleton a la classe ConnexionDB

(4) Créer une classe Test contenant la methode main.

Dans le main:

```
ConnexionDB c1 = ConnexionDB.getObject("u1","us1","p1");
ConnexionDB c2 = ConnexionDB.getObject("u2","us2","p2");
ConnexionDB c3 = ConnexionDB.getObject("u3","us3","p3");
```

afficher les attributs de c1, c2, c3.

Singleton:
- 1ère constructeurs private
- 2ème attribut static private son type: la classe
- 3ème une méthode public static getInstance

VIII - Interfaces:

GAB

Recharge mobile:

Banque

Operateur

```
interface Itraitement {
public boolean faireRecharge(String numTel, double mnt);
}
```
↳ Signature

```
class Traitement implements Itraitement {
public boolean faireRecharge(String numTel, double mnt) {
    // Traitement
}
}
```
↳ corps

Exemple:

```
Interface I1 {
    public void m1();   // toujours public et abstract
    public void m2();
}
~~Class~~ C1 implements I1 {
    public void m1() {...}
    public void m2() {...}
}
```

Exercice 1:

```
interface I1 {
    void m1(); // aucune erreur
    public void m2(); // Pas d'erreurs
    abstract ~~final~~ public void m3(); // erreur
    public void m4() {} // erreur
    ~~protected~~ void m5(); // erreur
}
```

```
class C1 implements I1 {
    public void m1() {} // erreur
    public void m2() {} // erreur
    public void m3() {} // pas d'erreur
    ~~protected~~ public void m5() {} // erreur
    public void m4() {} // Erreur due à l'absence du corps du m4()
```

Exercice 2:

```
Interface I2 { // pas d'erreurs
    void m1();
    void m2();
}
```

```
Interface I3 extends I2 {
    static final int age = 10; // on doit initialiser
    abstract public void m3();
}
```

```
class C2 extends C3 implements I3 { } // il faut donner les corps des m3, m1, m2
class C2 ~~implements I~~3 extends C3 { } // il faut commencer par extends puis implement
class C3 { }
```

Exercice TP:

Nous souhaitons comparer ~~[illegible]~~ les étudiants, les professeurs et les administrateurs. mais il faut donner un critère de comparaison.

→ les étudiants par leur âge

→ les professeurs par leur expérience

→ Les administrateurs par leur nom

1) Créer les classe étudiant, professeur, Administrateur avec leurs attributs

les Getters + Setters + le contructeurs

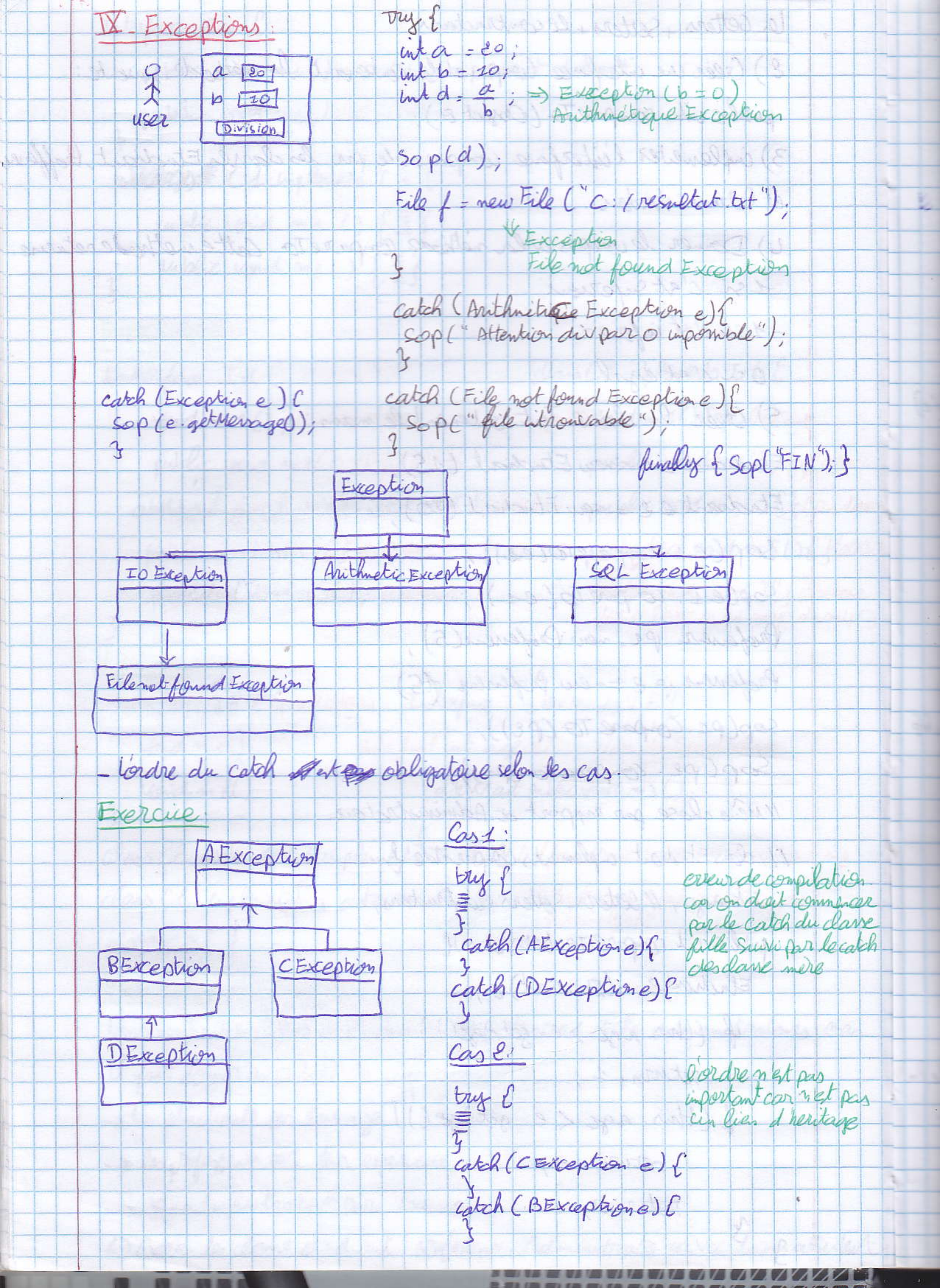
2) Créer une interface Comparable contenant la méthode suivante :
public int compareTo (Object o);

3) implementer l'interface comparable par les classes Etudiant, Proffeseur et Administrateur.

4) Donner le corps à la méthode compareTo. Cette methode returne :
1 si c'est superieur.
-1 si c'est inferieur
0 si c'est égal

5) Créer la classe Test contenant le main :

```
Etudiand e1 = new Etudiant (15);
Etudiant e2 = new Etudiant (20);
 sop(e1.compareTo(e2));
 sop(e1. compareTo(e1));
Profeseur p1 = new Profeseur(5);
Profeseur p2 = new Profeseur (15)
 sop(p1.CompareTo (p2));
 Sop(p2. compareTo (p1));
// Même chose par rapport à Administrateur.
// class Etudiant implements Comparable {
   int age; // Getters + Setters + contructeurs,
   public int compareTo (object o){
     Etudiant e = (Etudiant) o;
      if (this. age > e. getAge())
          return +1;
      if (this. age < e. getAge())
            return -1;
          return 0;
   }
}
```



IX - Exceptions :

user — a [20] b [10] [Division]

```
try {
  int a = 20;
  int b = 10;
  int d = a/b ;      => Exception (b = 0)
                        Arithmétique Exception
  sop(d);
  File f = new File ("C:/resultat.txt");
                     ↓ Exception
                       File not found Exception
}
catch (Arithmetique Exception e){
  sop("Attention div par 0 impossible");
}
catch (File not found Exception e){
  sop("file introuvable");
}
finally { Sop("FIN"); }
```

```
catch (Exception e) {
  sop(e.getMessage());
}
```

Exception → IO Exception, Arithmetic Exception, SQL Exception

IO Exception → File not found Exception

- l'ordre du catch est obligatoire selon les cas.

Exercice

AException → BException, CException

BException → DException

Cas 1 :

```
try {
  ≡
}
catch (AException e){
}
catch (DException e){
}
```

erreur de compilation car on doit commencer par le catch du classe fille suivi par le catch des classe mère

Cas 2 :

```
try {
  ≡
}
catch (CException e) {
}
catch (BException e){
}
```

l'ordre n'est pas important car n'est pas un lien d'heritage

Cas 3 : erreur

```
try {
  ...
}
catch(BException e){
}
catch(DException e){
}
```

Cas 4 : Aucune erreur.

```
try{
  ...
}
catch(CException e){
}
catch(AException e){
}
```

Exceptions personnalisée :

Dev 1 :

```
class Ville {
    int nbHabitants;

    Ville(int nbHabitants) throws NbNegatifException{
    if (nbHabitants >0)
        this.nbHabitants=nbHabitants;
    else
        throw new NbNegatifException();     // try(){ throw ....}
    }                                       // catch(){....}

class NbNegatifException extends Exception{....}
```

Dev 2 :

```
main ....
try {
    Ville v1 = new Ville(1000);
    Ville v2 = new Ville(-1000);
}
catch (NbNegatifException e) {
    sop("Attention Nb est negatif");
}
```

Exercice Exceptions personnalisée :

```
class Test {
-------------------------------------
..... m1 () { try { m2() ;}                  Boss
  catch (AException e) {...}
}
-------------------------------------
..... m2() throws AException {           Presentation
          m3();
      }
-------------------------------------
..... m3() throws AException {           RG
          m4();
}
-------------------------------------
```

m 4 () throws AException {

}   BD

TP Exercice 1 :

① Créer une classe Test contenant la méthode main

② Créer deux variable a et b de type String dans main

③ Convertir a et b vers int en utilisant.

int ia = Integer.parseInt(a) ;

④ Affecter à a et b les valeurs

cas 1 : a = "10" b = "2"   cas 2 : a = "10" b = "m"   cas 3 : a = "10" b = "0" (faire la a/b)

⑤ Gérer les exceptions issues des cas 2 et 3

⑥ Ajouter un bloc catch pour Exception

⑦ Changer l'ordre des bloc catch ⇒ Votre remarque

⑧ - Afficher le message "FIN" ∀ l'exécution de votre programme Avec ou Sans Exception.

TP Exercice 2 :

Créer une classe Test2 contenant les 4 méthode de l'exercice Exceptions personnalisé

TP Exercice 3 :

Exemple Exceptions personnalisé.

Correction du contrôle :

Question du cours :

1) Une classe , Un objet

2) le role et les caractéristiques des constructeurs

3) Le tableau

4) final, static, super(), super. : appeller les attributs ou méthodes de la classe mère.

5) Polymorphisme : c'est le fait de créer des méthodes avec une classe mère comme paramètre et l'appeler par des objets des classes filles.

Redéfinition.

6) les classes abstraites et "abstract"

Exercice 1 :

```
1) Class Version {
     Private Double idVersion ;
     Private Date dateVersion ;
     Private String CommentaireVersion ;
     Private int nombremodifs ;
3) Public Version (Double idVersion) {
          this.idVersion = idVersion ; }
   Public version (Date dateVersion, string commentaireVersion, int nombre-
       modifs) {
          this.dateVersion = dateVersion ;
          this.CommentaireVersion = CommentaireVersion ;
          this.nombreModifs = nombremodifs ;
     }
2) Public Double getIdVersion () { return this.idVersion ; }
   Public void setIdVersion ( Double idVersion) { this.idVersion = idVersion;}
   Public Date getDateVersion() { return this.dateVersion ;}
   Public void setDateVersion(Date dateVersion) { this.dateVersion = dateVersion;}
   Public String getCommentaireVersion() { return this.commentaireVersion ;}
   Public void setCommentaireVersion(String CommentaireVersion ) {
          this.commentaireVersion = commentaireVersion ; }
   Public int getNombremodifs () { return this.nombremodifs ; }
   Public Void setNombremodifs(int nombremodifs ) {
          this.nombremodifs = nombremodifs ; }
4) Public Static void main ( String [] args ) {
          Version V1 = new Version( 5678934 ) ;
          Version V2 = new Version( new Date("05/03/2015"), "Version new", 6) ; }}
```

Exercice 3 :

```
class Usine {
```

```
Private Static Usine U;
 private Usine(){ }
  Public Static Usine getObject(){
    if (U == null)   u = new Usine();
/*else*/ return u;
 }
}
```

Exercice 2 :   code 1 :

Doc: ~~static~~ public void m() {...}   affichage ?

Livre: ~~Static~~ public void m() {...}

sr

affiche (methode object)

Livre
Livre

(methode de classe)

document
Livre

```
Document d1 = new ~~Document()~~ Livre();
Livre d2 = new Livre();
   d1.m();
   d2.m();
```

code 2 :

Sport
1 2 1
True

X. Collections :

1) Tableaux :

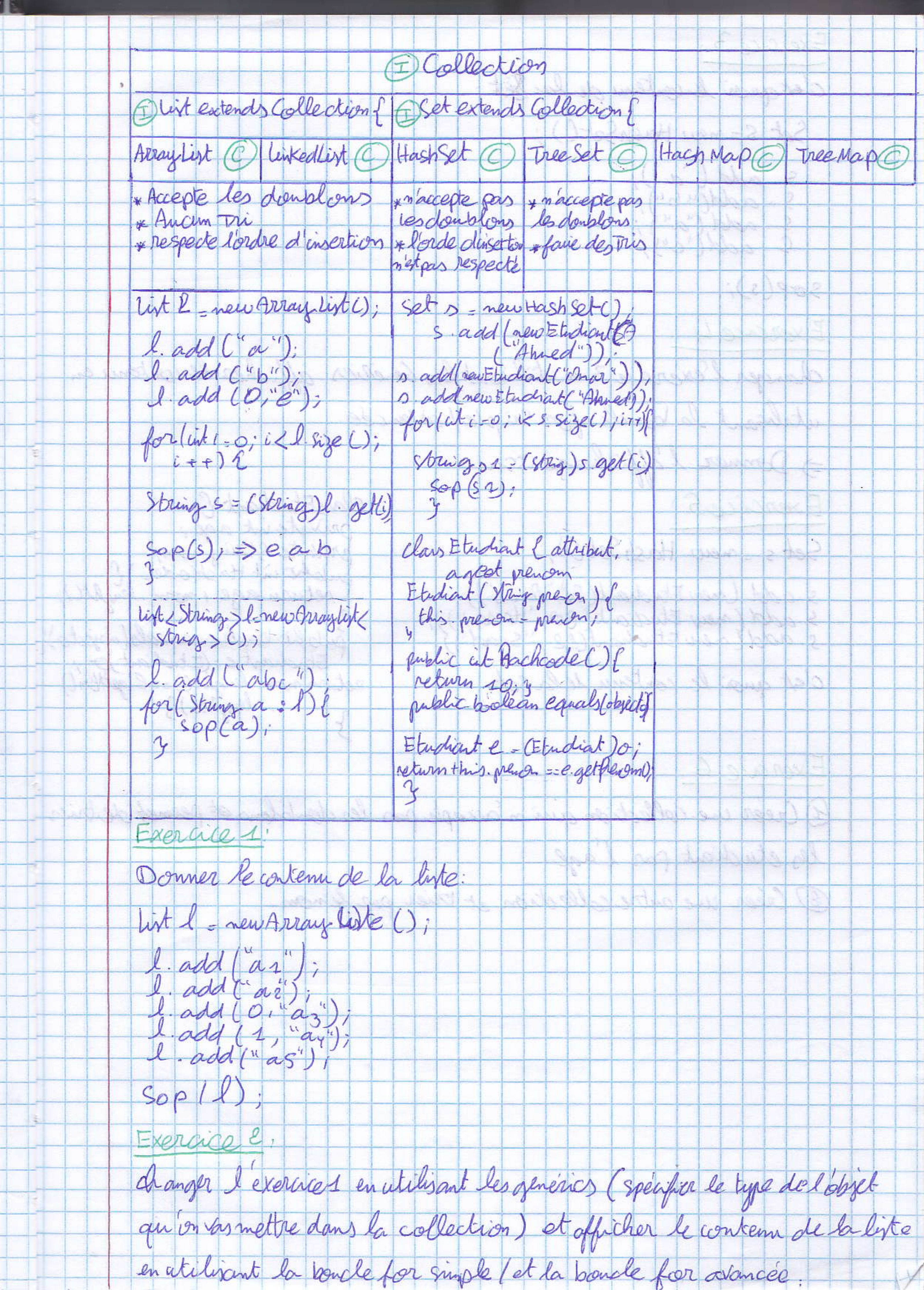
- ```
  String[] monTab = new String [3];
  monTab[0] = "abc1";
  monTab[1] = "abc2";
  monTab[2] = "abc3";
  ```
  ⟺ `String[] monTab = {"abc1", "abc2", "abc3"};`

- `String[] monTab ;` ⟺ `String monTab[ ];`
- les boucles :
  ```
  for(int i = 0 ; i < monTab.lenght ; i++){
      Sop(monTab[i]);
  }
  ```

① Il faut pas que ce tableau contient de doublons.

② Il faut trier les elements de mon tableau.

il y a 3 types de Collection.
  ```



(I) Collection					
(I) List extends Collection {		(I) Set extends Collection {			
ArrayList (C)	LinkedList (C)	HashSet (C)	TreeSet (C)	HashMap (C)	TreeMap (C)
* Accepte les doublons * Aucun Tri * respecte l'ordre d'insertion		* n'accepte pas les doublons * l'ordre d'insertion n'est pas respecté	* n'accepte pas les doublons * faire des Tris		

List l = new ArrayList();
l.add("a");
l.add("b");
l.add(0,"e");
for(int i=0; i<l.size();
i++){
String s = (String)l.get(i)
sop(s); => e a b
}
List<String> l=new ArrayList<String>();
l.add("abc")
for(String a : l){
sop(a);
}
		Set s = new HashSet();
		s.add(new Etudiant(("Ahmed"));
		s.add(new Etudiant("Omar"));
		s.add(new Etudiant("Ahmed"));
		for(int i=0; i<s.size();i++){
		String s1 = (String)s.get(i)
		sop(s1);
		}
		class Etudiant { attribut,
		a get prenom
		Etudiant(String prenom){
		this.prenom = prenom;
		}
		public int hashcode(){
		return 10;}
		public boolean equals(object o){
		Etudiant e = (Etudiant)o;
		return this.prenom == e.getPrenom();
		}			

Exercice 1 :

Donner le contenu de la liste :

```
List l = new ArrayListe();
  l.add("a1");
  l.add("a2");
  l.add(0,"a3");
  l.add(1, "a4");
  l.add("a5");
Sop(l);
```

Exercice 2 :

changer l'exercice 1 en utilisant les génériques (spécifier le type de l'objet qu'on va mettre dans la collection) et afficher le contenu de la liste en utilisant la boucle for simple / et la boucle for avancée :

Exercice 3 :

c'est quoi le contenu de la Set :

```
Set s = new HashSet();
  s.add("a");
  s.add("b");
  s.add("a");
  s.add("c");

sop(s);
```

Exercice 4 :

changer l'exercice 3 en utilisant les Generics. afficher le contenu en utilisant la boucle for simple / avancée.

=> Donner l'affichage issu.

Exercice 5 :

```
Set s = new HashSet();

s.add(new Etudiant(22, "Omar"));
s.add(new Etudiant(20, "Hassan"));
s.add(new Etudiant(22, "Saad"));
```

c'est quoi le contenu de la set

```
class Etudiant {
  private int age;
  private String nom;
  public int HashCode() {
    return age + nom.lenght();
  }
  public boolean equals(object o) {
    Etudiant e = (Etudiant) o;
    return nom.lenght() == e.getNom()
      .lenght() }
}
```

Exercice 6 :

① Créer une collection qui n'accepte pas les doublons et permet de trier les etudiant par l'age.

② Créer une autre collection et trier par le nom.